CICERO RD38: A Distributed Information System for HEP Controls

J-M Le Goff
CERN, Geneva, Switzerland

R McClatchey
Dept. of Computing, Univ. West of England, Bristol, UK

Abstract

Large scale, distributed industrial and scientific systems share many technical problems in the implementation of their control systems. With the advent of standards in distributed object technology, it is now feasible to reuse control objects across control systems. The CERN CICERO project aims to use object-oriented methods to design the main building blocks of a generic control information system based on the distributed object standard, CORBA [1]. CICERO is producing an integrating environment (Cortex [2]) into which distributed user control objects will ultimately be 'plugged and played' and a supporting information system for the configuration and management of that environment. Cortex has been designed to be sufficiently generic in nature to allow its reuse by any future control system used at CERN and potentially by medium to large scale industrial control systems such as power systems, satellite control and telecommunications network management. This paper describes the main design concepts behind Cortex and the enabling technology and the software engineering methods used in implementing Cortex.

1 Introduction

The latest experiments in particle physics at the CERN laboratory in Geneva are composed of large numbers of sophisticated detectors and devices each potentially constructed by geographically separated development teams using disparate electronics and software. Each detector requires software to control the acquisition of data and the supervision and operation of great numbers of sensors and actuators. Control activities include monitoring devices, maintaining the safety of the experimental setup and the control of low level automation loops to automatically maintain devices in operational conditions. To reduce development costs physicists are looking for partially reusable solutions to their technical problems such as the incorporation of industrial products with home-grown products [3] for tasks in the control system. However, in the recent past it has proved difficult to integrate commercial products together (e.g. PLCs with VME) and to integrate these products with existing CERN-made control (sub-)systems. In essence, what is required is an overall framework to facilitate integration between control system elements. Such a framework (or software integration platform) should go beyond defining standard interfaces, it should guarantee that commercial products can exchange information and collaborate regardless of the organisation of the overall control system.

The next generation of CERN experiments at LHC - the Large Hadron Collider [4] - will involve collaborations of many tens of institutes and over 1,000 physicists, engineers and computer scientists from around the world. The knowledge required to construct and monitor the experimental (sub-) detectors will be distributed between these institutes making it difficult to impose standards. There will consequently be significant problems of information transfer to ensure that each (sub-) detector retains autonomy of control but can work with other (sub-) detectors for data-acquisition. In addition, the LHC detectors will be required to have a long lifecycle since the experiments will take data for several years and, as a consequence, maintainability will be an important consideration. The experimental groups will also be working to very tight time and cost schedules. As the experiments grow, so the control system should grow from an initial lab-based test system to test-beam operation and to the fully-fledged experimental system. The Cortex element of the CICERO project intends to provide an integrating scheme (Cortex [2]) to enable the building of distributed control systems where

responsibilities are distributed amongst control elements that have to collaborate together.

Experience of the development of control systems for the LEP experiments [5] and anticipation of the increased demands which will be generated by the new and larger experiments for LHC has identified the main constraints for the design of such an integrating scheme. Firstly, since the LHC experiments will be equipped with > 100,000 sensors and activators, the **management of control system complexity** is a major consideration. Object-oriented design techniques embodying abstraction, inheritance and the use of classes and objects will aid the design here. Secondly the problem of **concurrent and collaborative software engineering** requires addressing. This is particularly true when the development of the control software is carried out by engineers who are separated geographically. Thirdly, the software developed should provide **stability, flexibility and availability** of control system elements so that system down-time (such as that for upgrades) is minimised. Finally, the control system software should provide **balanced, distributed processing** in the heterogeneous environment of High Energy Physics (HEP) control systems.

The next section identifies the characteristics of control systems as used in HEP and illustrates how the concepts of *hierarchy* and *collaboration* are important in control. The design philosophy behind the construction of Cortex is identified in the sections that follow showing how those design constraints identified above have been addressed. The enabling object-based technology and standards used in the design and development of Cortex are later investigated before the current status and future development plans for Cortex are described.

2 Control Systems Structure in High Energy Physics

From the point of view of control, a HEP accelerator or experiment can be visualised as a set of devices (muon detector, electromagnetic calorimeter, etc.) and systems (data acquisition system, gas system, etc.) performing specific functions and collaborating, by the exchange of control information, in order to achieve the accelerators's or experiment's objectives. Each of these devices and systems, are usually decomposed into subsystems, with specific functions, different information needs and different degrees of autonomy. These subsystems could be further subdivided, to the appropriate level of granularity required by the underlying hardware or by functional decomposition (Figure 1).

In order to properly specify the collaborative distributed control system, two basic features are required to be offered to the users of the control system. On the one hand, the users need to be able to organise hierarchically the different elements of the collaborative distributed control system. On the other hand, the user must be able to specify the collaboration needs between these elements, independently of their hierarchical organisation.

In the first case, the hierarchical organisation of the control system reflects the layout of the experimental setup, supports detector-specific global operations (e.g. for data taking or beam operation) and local operations (for calibration and test purposes e.g. ramp-up high voltage) and provides a representation which allows apparent complexity of the experiment to be hidden from the user. In the second case, the collaboration aspects provides the user with facilities to specify which information is to be shared between control elements and how that information is to be shared. This includes definition of the formats of shared data or services and the form of transport of the data. Two major mechanisms for data transfer and service invocation are required: *multicast communication* which can be used for functions such as start calibration or the distribution of monitoring data from a single producer to multiple consumers and *point-to-point communication* which can be used for interlocking front-ends or the dispatch of a command for invocation between a requester and a performer (e.g. operator asks the muon system to ramp down its high voltage to 2 KV).

All the above needs can be abstracted by two orthogonal relationships which must be satisfied by any system used for controls in HEP:

• the “composition” relationship which describes the hierarchical organisation of control elements to support global behaviour. For instance, the “Experiment” element is composed of the “Muon detector”, “Run control”, “Gas system”, “Cooling and ventilation”, “Security”, “Tracker”, “ECAL”, and “HCAL” elements and the “Muon Detector” itself is composed of “High Voltage”, “Low Voltage”, “Discriminator System”, “TDC” and “Alignment System” elements and

• the “collaboration” relationship which is the hierarchy independent communication between elements to support dedicated information and command exchanges. For instance, the “Muon detector” element is collaborating with the “Gas system” element.

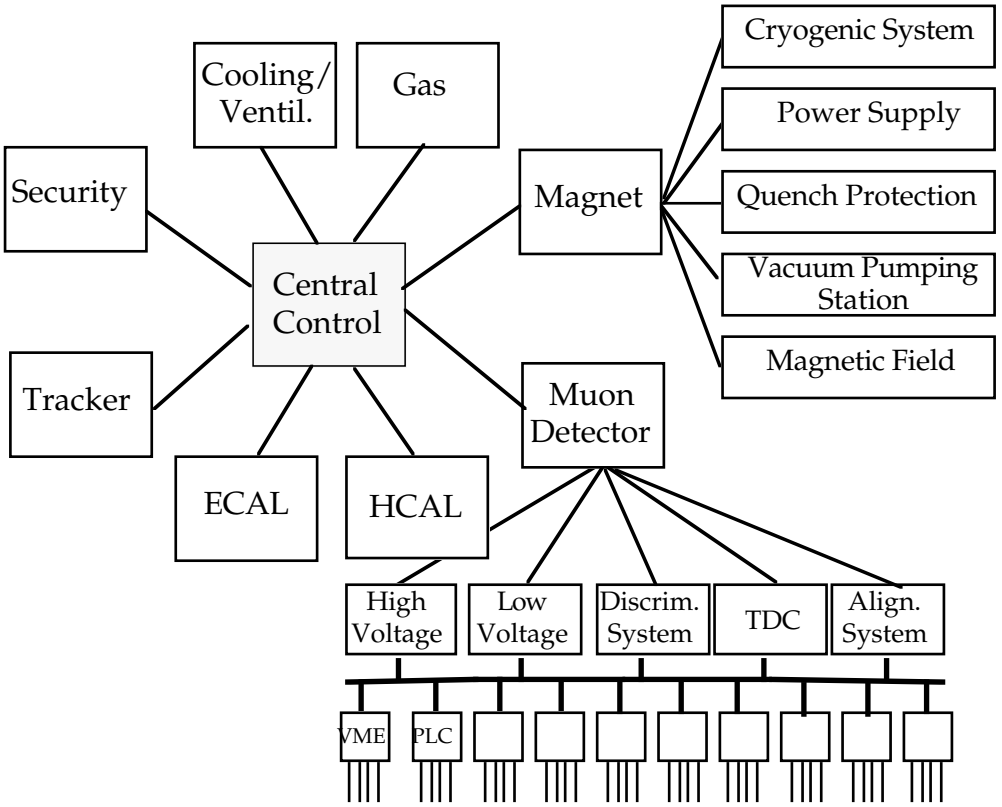

Figure 1: An experiment hierarchy

3 Scope and Constraints in Cortex

HEP experiments and accelerators are normally developed by various teams with specific task assignments. Usually, the different parts of a HEP experiment are developed by different institutes in their laboratories and gathered together later after being tested. The experiments are tuned up at CERN before new tests, or calibration and operation. For instance, the gas system and the high voltage system of a muon detector may be produced and tested by two different teams. Cortex must support an equivalent life cycle for the collaborative distributed control systems of these experiments or accelerators. In particular, it should be possible to develop control systems and later to integrate them in a collaborative distributed control system without major code modifications. Users should only have to configure the existing collaborative control system, by introducing their part of the development and specifying the information and commands to be exchanged with this part in the global control system.

Although the different elements of a HEP experiment are operated globally during data taking, sometimes some subsystems or devices have to be operational while some others are not. For example, during calibration and test, the muon detector and the gas system can be operational while the magnet is switched off. Cortex shall promote the modularity of distributed control systems in order to allow some integrated control systems to be operational while some others are not. Malfunctioning hardware may lead control elements to send incorrect information to other elements, which can consequently take inadequate actions. If the hardware cannot be repaired immediately, Cortex shall allow operators to prevent this incorrect information from being broadcasted. Cortex will also support users during the reengineering phases of the integrated control systems by allowing them to reuse and integrate existing software.

Cortex is required to support the entire collaborative distributed control system life cycle by:

• providing a multi user development environment,

• supporting the testing and simulation of users' control elements with respect to the integration within the Cortex control system,

• offering tools to operate and protect the control system from faulty processes implementing some control elements,

• allowing the integration of already existing control systems,

• allowing collaboration with other autonomous control systems.

Cortex intends to provide a control system designer with the ability to integrate his particular control system efficiently and with minimal cost and effort.

The architecture of any distributed control system will change during the lifetime of the accelerator or the experiment it is operating. The control system integrating platform, Cortex, must therefore support a mechanism to allow a new version of the distributed control system to be in preparation while an older one is operated on-line. It shall also support the backup and the restore of a given version of the collaborative distributed control system. Furthermore, tests and validation (and possibly simulations) of a new configuration will be needed before it is applied to the operating on-line system.

4 The Cortex Design Philosophy

• The above constraints have led to a so-called dual-face approach (see figure 2.) being taken in Cortex:

• an off-line Cortex representation is required to handle the logical descriptions of the architecture of the distributed control system and to describe the various information and commands to be exchanged between the different control elements. This so-called *Repository* also holds the description of the hardware model from which the on-line distributed control system is constructed, and

• an on-line Cortex representation is also required through which the control elements can exchange information and commands in a pseudo- 'plug-and-play' fashion. Control elements operating within the Cortex Infrastructure can access the Cortex Repository through this so-called *Infrastructure.* A generation mechanism is provided to facilitate updates of the on-line Infrastructure according to the Repository contents.

The responsibility of Cortex is twofold: on the one hand it has to support the description of the architecture of the distributed control system and the definition of the information and commands to be exchanged between the control elements. The off-line representation of the control system therefore addresses the issues of the **management of control system complexity** and that of **concurrent and collaborative software engineering** identified earlier. On the other hand, Cortex must transport and distribute these data and commands to the appropriate

control elements when part or all of the distributed control system is in operation. This distribution must be independent of the number of hardware elements on which the various control elements are operating. The integrating framework must be flexible enough to support the addition or removal of control elements, without deteriorating the operation of the rest of the distributed control system. The on-line Cortex Infrastructure thereby addresses the demands of **stability, flexibility and availability** of control system elements and that of providing **balanced and distributed processing** for the control system.

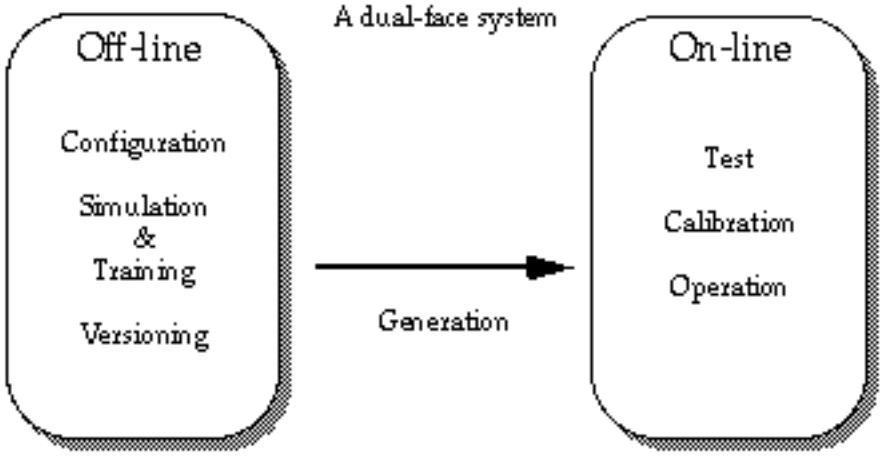

Figure 2: The dual-face approach

5 Cortex off-line

5.1 Components

Off-line a Cortex control system is visualised as a collection of collaborating components. These components map onto those on-line processes in a control system which produce or consume information and they can be of various kinds. They may or may not be “control components”. For example gas systems or high voltage systems are managed by software that are control components. Each has a read-out system and has responsibility for the hardware. Indeed some control components may need real time capabilities, for instance, elements interfacing front-ends performing interlocks. On the other hand, an on-line documentation element or a user interface element are “non-control elements” - they do not have responsibility for the hardware. They may, however, support high level control functionalities such as alarm filtering, user assistance, preventative maintenance etc. Cortex can therefore be used to integrate software which implements facilities common to any kind of control element such as loggers, archivers, retrievers, DUIs, final state machines, etc.

5.2 Compositeness and Collaborative Groups

Compositeness is the mechanism proposed in Cortex to support the logical encapsulation of a distributed control system. Components may be composed of smaller components. Such components are referred to as composite components and are often used to separate functions or to provide the granularity required by the underlying hardware (figure 3). Users must be able to operate the complete control system from a global standpoint or operate each component independently via the composite components. Additionally, users must be able to specify communication requirements at any level of component, regardless of the inherent hierarchical organisation. In practice then a composite component may be chosen either:

• to represent a global control process when it will support global operations such as those to start/stop the global control process or

• to represent a local control process when it will support operations such as start/stop for any specific encapsulated component.

Grouping is a complementary mechanism to Compositeness proposed in Cortex to allow specification of information and command exchange at any level of granularity. A collaboration group is composed of a set of components that make available certain information and services to the other components in the group. Two components will be able to exchange information and commands if and only if they belong to the same collaboration group. Components can be part of more than one group. Collaboration groups can be established across encapsulations of sub-systems. The combination of compositeness and collaboration groups allows the user to refine and optimise the communication at an appropriate level of control system component.

5.3 Publishing and Subscribing

Within a collaboration group, a component can provide information to other components by publishing items (data or services). If granted permission by the publisher, any component of a collaboration group (other than the publisher) can access this information by subscribing to the published items. The set of published and subscribed items handled by a component within a collaboration group is called a component interface. A component usually has a different interface for each collaboration group in which it is a member.

A component can subscribe in one collaboration group and republish in another collaboration group. The publish and subscribe mechanisms are fully asynchronous. In particular, an item can be depublished while it is still subscribed by other components. If this situation is propagated to the on-line system, the corresponding on-line Infrastructure will support such incon-

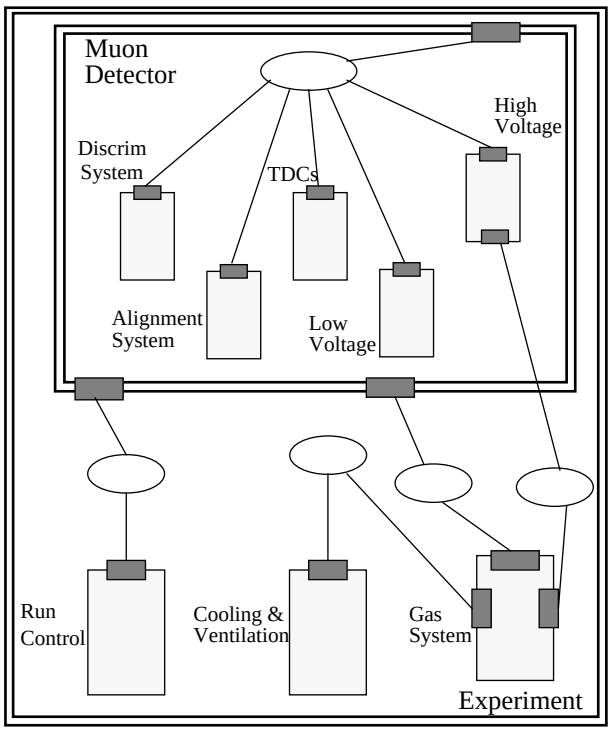

Figure 3: (Composite) Components & Grouping

Key: Groups are represented by ellipses, Components by rectangles and Component interfaces by shaded boxes. Composite Components are shown as double rectangles.

sistencies and inform the appropriate subscriber components. Compositeness and collaboration grouping are the mechanisms for supporting information abstraction in Cortex: some of the collaboration groups can be organised to enforce encapsulation of sub-systems (so-called *strict encapsulation*). This will allow composite components to subscribe within their encapsulation and publish refined information to other composite components in other collaboration groups.

6 Cortex on-line

The Cortex on-line Infrastructure is a set of entities responsible for the distribution of information and for the transmission of commands (service invocation) to the appropriate components, according to the Cortex Repository specifications.

The Cortex Infrastructure provides two selectable ways of data exchange between components. The *push mechanism* which allows components to push new information into the Infrastructure or to receive information from the Infrastructure and the *pull consumer mechanism* which allows components to retrieve information from the Infrastructure at their convenience. The push mechanism is recommended for security information such as alarms. The pull mechanism is more suitable for monitoring components offering refreshed information upon users' request. In both cases, only information specified in the Cortex Repository will be transported and delivered to the appropriate components. Version inconsistency between the information sender and the information receiver are handled by the Infrastructure at message level. For example, a component may pull from the Infrastructure items which are no longer published (in the Repository). The dynamic information contained in these items is no longer refreshed and the corresponding component will be informed. Time stamps will contain the last time these items have ben refreshed.

The on-line architecture supports a separated set of messages to handle service invocation called command messages. Commands are persistent in the Infrastructure from the moment they have been issued until they are completed (successfully or not) or refused by performers. Two basic types of services are available. Firstly services can be cancellable or non-cancellable: a requesting component can cancel its request while the component offering the service is processing the command. Secondly services can be multi- or single-requestable: more than one requesting components can issue a command to the same performer component.

More than one requesting component can invoke the same single-requestable command hence addressing the same performing component. The on-line architecture handles possible access conflicts using an internal protocol based on locking. Commands are not direct implementations of operations in the OMG Object Model, which do not support some specific control functionalities such as authentication, availability and progress report features.

7 Enabling Technology

In an effort to maximise the reusability of code in Cortex and to provide an incremental development route, object standards have been followed in developing Cortex. In particular the Cortex on-line Infrastructure is based on the Object Management Architecture (OMA) and the Cortex off-line Repository on the Object Database Management Group (ODMG) standard. The following sections introduce these standards and identify the important aspects of the standard-gauge Cortex. The use of the OMA standards are considered in the next section of this paper where reusability of control system components is considered.

7.1 The OMG Object Management Architecture Standard

The Object Management Group (OMG) is an industry consortium dedicated to creating object management standards necessary to achieve the goal of interoperability between heterogene-

ous, distributed object based systems. The Object Management Architecture (OMA) [6] is the reference model for the OMG standards and identifies three major categories of specifications or architectures:

• Common Object Request Broker Architecture (CORBA)[7]

• Common Object Services Specification (COSS)[8]

• Common Facilities Architecture (CFA)[9]

In the OMA object model, objects provide services to clients as shown in figure 4. An interface is a description of a set of the possible services that a client may request and is specified in Interface Definition Language (IDL). Clients are not written in IDL but in the implementation language for which mappings have been provided. IDL mappings are currently available for C and C++.

The initial focus of the OMG was on the specification of the Object Request Broker (ORB) as detailed in the Common Object Request Broker Architecture (CORBA) document. The ORB forms the key part of the architecture by describing how objects are located, operations accessed and arguments passed transparently and flexibly such that integration of a variety of distributed object based systems can be achieved. CORBA specifies IDL, the object invocation interfaces together with an interface Repository on the client side, object adaptors (OA) on the server side and the ORB core. It is the ORB core which does the object location, message delivery and binding between clients and object implementations. There are two ways that a client can make an object invocation request, one is through a static interface and the other through a dynamic interface. Whichever invocation interface is used a target object cannot tell which method was used. The important difference between the invocation interfaces will be the amount of time taken to invoke the implementation object.

The actual object implementations are constructed from programs which could be executable scripts or loadable modules. An implementation object can be designed so that a single program implements the whole objects interface behaviour or one program can be used to implement each of the methods. The method binding interface to the ORB core is through skeletons. However the object implementation interacts with the ORB core in a variety of ways, for example, to register itself, to request an object reference and to invoke ORB serv-

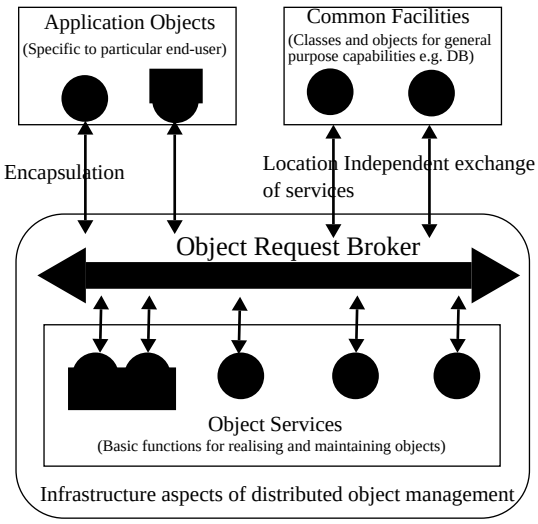

Figure 4: The OMG/CORBA Model for CICERO.
Application objects: Cortex Infrastructure objects.
Common facilities: OODBMS, Real-Time controls

ices. This interface is provided by the Object Adapter (OA). The OA defines most of the services from the ORB core that the Object Implementation can depend on such as activation, deactivation and access control to object implementations. Different ORBs and different operating environments will provide different services and levels of service, if the ORB service is available then the OA simply provides an interface to it and if the service is not available then the OA must provide it. The CORBA specification suggests that it is not necessary for all object adaptors to provide the same interface or functionality an example of a special OA interface would be one that connects to objects stored in an object-oriented database.There is one OA that all CORBA implementations must provide and that is the Basic Object Adapter (BOA). The BOA is concerned with activation, deactivation and access control of object implementations and thus the main responsibilities are:

• Generation and interpretation of object references

• Authentication of the principal making the call

• Activation and deactivation of the implementation

• Activation and deactivation of individual objects

CORBA provides the basic communication channel through which objects interact however the system services it provides are limited. Fundamental system services such as naming, replication, security, transactions and time are key to building distributed applications. The Common Object Services Specification (COSS) defines in terms of interfaces and objects a collection of these services. COSS volume 1 covers Naming which provides the ability to attach textual names to object references, Event Notification which provides an event notification service for unexpected events, Life Cycle Support for creating, deleting, copying and moving of objects and a Persistent Object Service. COSS volume two which is expected in mid 1995 will specify transactions, externalisation, relationships and concurrency services. COSS Volume 3 will feature Security and Secure Time Service and COSS Volume 4 will include Object Query and Object Properties services. These later two volumes are expected later in 1995.

The Common Facilities Architecture (CFA) is the third and final area of OMA to be defined (interfaces and objects) and is a collection of higher level services which may be broadly applicable to many applications. Four major horizontal domains for such facilities have been identified so far: User Interface, Systems Management, Information Management which covers the modelling, definition, storage, retrieval, management and interchange of information, and Task Management which covers the automation of work. Vertical Market Facilities represents technology that supports various vertical markets such as retailing, telecoms, CAD or health care. Each Common Facility interface is defined in IDL and as such may inherit behaviour of the more fundamental Common Object Services. Similarly implementers building distributed applications can make extensive reuse of Object Services and Common Facilities by OMG IDL based inheritance.

7.2 The Object Database Standard (ODMG)

The Object Database Management Group (ODMG) has put forward a set of standards allowing an Object Database Management System (ODBMS) user to write portable applications, i.e. applications that could run in more than one ODBMS product (the schema will be portable as well as the application accessing it). The proposed standard will, eventually, be helpful in allowing interoperability between different ODBMS products, allowing the development of distributed heterogeneous database communicating through the OMG Object Request Broker.

The ODMG defines an ODBMS to be a Data Base Management System (DBMS) that integrates database capabilities with object-oriented programming language capabilities. An ODBMS makes database objects appear as programming language objects, extending the language with transparently persistent data, concurrency control, data recovery, associative que-

ries and other database capabilities. The major components of an ODBMS are the following:

• the Object Model. This is based on the OMG Object Model. An ODBMS Profile has been created as an extension of the OMG Core Object Model to support the ODBMS specific needs (e.g. Relationships),

• the Object Definition Language (ODL). This is the data definition language for the ODBMSs. It is based on the OMG Interface Definition Language,

• the Object Query Language (OQL). This is a declarative language for querying and updating database objects. It is based on the relational standard SQL,

• the C++ Language Binding. This explains how to write portable C++ code that manipulates persistent objects. It is called the C++ Object Manipulation Language. The binding also includes a version of the ODL that uses the C++ syntax, a mechanism to invoke OQL and procedures for operation on database and transactions.

• the SmallTalk Language Binding (in preparation).

The current version of the standard is ODMG-93, ODMG-95 is in a draft version. ODMG-93 foresees the ability to access an ODMG-compliant product through an ORB using a special Object Adapter for this ODMG product. Users have to design this adapter or intermediate objects that can make use for instance, of C++ OML and of the OMG C++ Language Mapping.

7.3 Mapping Enabling Technology onto the Cortex Design Philosophy

In Cortex, an ODBMS is being used as the vehicle for the off-line Repository to support a standardised access for the CORBA objects in the on-line Infrastructure. The Repository has been designed to support the notions of Compositeness and Collaboration Groups, Publishing and Subscription and Components as described in an earlier section. ODBMSs provide persistence of object information, and all the advantages of DBMS systems such as version management, concurrency control, security and recovery. This enables control system designers to save a full description of the experimental setup in an object base and to modify that description over time as the experiment grows.

The Cortex on-line Infrastructure is instantiated as a set of CORBA objects responsible for the distribution of information and for the transmission of commands to the appropriate components, according to the description resident in the Cortex Repository. On-line objects in Cortex are written in C++ and use Iona Technologies implementation of CORBA, called Orbix, for object location, access and communication services. The physical location of the components and the Infrastructure will depend on the hardware setup available. This setup may evolve with time for performance reasons or for maintenance purposes. In these cases, the system functionalities must be maintained when part of the hardware is changing. This operation should take place without disturbing the operation of the parts of the distributed control system which are not involved in this upgrade. The location transparency is fully supported by CORBA. CORBA makes no provisions for message sender identification. Any program can potentially send a message to a CORBA object. To avoid unpredictable overloads, Cortex provides an authentication mechanism to ensure that for any new starting process is effectively representing a component known to the Cortex Repository.

As an example of the use of the OMA standard in Cortex the next section investigates how Cortex provides reusability both of control system components and complete control sub-systems.

7.4 Reusability and Cortex

The basis for re-use in the CICERO project is through the use of CORBA IDLs and ORBs. Reusability can be exploited in CICERO both at a level internal to components and at a level

external to components. Firstly at the internal level, consider the re-use of component code as the hardware of the control system is evolving. By using IDL stubs for client invocation and an IDL skeleton to package the existing (component) code for CORBA compliance, the control system can be allowed to grow whilst reusing existing components. As an example, consider the incorporation into a UNIX-based control system of an existing data logger which runs on VMS/ORACLE. To reuse this component it is necessary only to build an IDL interface to this logger, using an ORB which supports VMS. Then a UNIX component will be able to send messages such as store and retrieve through this interface to log information without having to know the complexities of UNIX-VMS translation. In addition, such a component will be able to log any additional Cortex messages issued by any other existing components according to the description stored in the Cortex Repository.

Using CORBA IDL for interface specification permits the sub-division of a large software module into smaller, easier to manage units with simpler functionality. This facilitates reusability in that it supports:

• the evolution and partial upgrade of complex control systems and

• the re-use of existing (legacy) systems through CORBA objects.

In this example of reusability, partial re-engineering of component code is again required since the IDL specification takes place inside the component code.

CICERO is also able to exploit reusability at a level external to components. This can be achieved through the use of so-called Reusable Components. These can be regarded as templates for components with logical input/output which can be instantiated as many times as there are physical devices. At the time of instantiation, there may be no hardware assignment - only at the time of assignment will the desired functionality become apparent and the component code reused. For example, consider a 16-channel Analogue to digital Converter (ADC) read-out component coupled with an ADC Converter component. The first component is hardware independent, except for the ADC gain, and can be reused as many times as ADCs are needed in the system. The second component is context dependent and can evolve with the hardware of the system. If the second component is data driven and obtains its configuration data from the Repository, then the ADC/ADC Converter pair can be reused with no code modification in any subsystem of the experimental setup. Cortex offers the possibility of decomposing a complex control system into data acquisition components, command components and automation loop components offering the possibility of reusing or sharing components in different control systems.

Another example of reusable components is possible in CICERO when whole sets of components are reused at the control system level. Here, consider two Cortex control systems developed independently and merged at a point in time. For example, a development or test setup which has been locally setup and then physically moved to be made part of (integrated with) a larger control environment. In this case since there is compliance at the Cortex level, no code modification whatsoever is required and the integration takes place through the addition of an intermediate component which resolves the match between the two Cortex systems. That is, the intermediate component subscribes to the items of the test control system and republishes the converted items for the larger control system. In addition, it is possible to split the two control systems for independent running at a later point in time without code modification.

8 Software Engineering Standards in the Development of Cortex.

The previous sections noted the object-based standards adhered to in the development of Cortex and investigated reuse of control system components based on these standards. This section describes the methods that were followed in implementing Cortex and in particular concentrates on the software engineering techniques used which enabled the development of Cortex.

8.1 The ESA PSS-05 Standard

As HEP systems become more complex, the need for rigor in software engineering increases in importance. In addition, as the development of these complex systems is increasingly carried out remotely from CERN or by developers on short-term contracts at CERN, the need for clearly defined deliverables and interfaces between (sub-)systems also becomes crucial. As a consequence, software engineering standards have been investigated in the last few years by large groups of developers in HEP. Work at the European Southern Observatory [10] and the European Space Operation Centre [11] into standards has recommended the use of the European Space Agency's software engineering standards alongside those from the IEEE and IEE.

The European Space Agency Procedures, Specifications and Standards [12] method is an essential feature of CICERO. This standard has been developed for the European Space Agency to ensure that any project has the best chances of a successful outcome. There are two major parts: the products themselves and the procedures to produce them. The products are the documents and software used to create, use and maintain software. The procedures guide the system developer in project management, software configuration management, software verification and validation and software quality assurance.The documentation for this standard includes both mandatory and optional sections, divided into three levels. In order to meet the ESA standard, all mandatory operations must be carried out or documentation produced as appropriate. The process of production is divided into six phases, following the standard waterfall life-cycle model of User Requirements Definition, Software Requirements Definition, Architectural Design, Detailed Design and Production, Transfer and Operations & Maintenance. In addition there are documents on Structured Analysis, Fortran Coding, Ada Coding & C Coding standards.

Management of the software lifecycle is catered for in the ESA standards through the use of a Software Configuration Management Plan (SCMP), a Software Verification and Validation Plan (SVVP) and through Software Quality Assurance (SQA). These plans are detailed in [12] and provide the project manager with requirements for identifying, controlling, releasing and changing software releases and for recording their status. The SVVP provides for the review, testing and auditing of the delivered software products. Further, the project manager can ensure quality is being maintained in software delivery by following the recommendation of the SQA plan.

The ESA standards were originally based on the ‘Waterfall Model’ of the software lifecycle, following a phased approach to software development. Modified forms of this approach include the incremental delivery and evolutionary development models. In the development of Cortex an *evolutionary* approach has been adopted which overcomes some of the limitations of the waterfall model. The evolutionary approach allows for the planned delivery of multiple releases of Cortex, with each release incorporating the experiences of earlier releases in a manner analogous to the ‘Spiral Model’ of software development suggested by Boehm [13].

8.2 The OMT Methodology

The CICERO project has selected the European Space Agency standard PSS-05-01 [12] as the life-cycle model to support the development of its software. Within that framework more specific software engineering methods are being used. Foremost amongst these is the object-oriented design method developed by James Rumbaugh and colleagues, OMT [14].

The OMT method comprises a number of models which are developed and enhanced as the project moves from requirements analysis through design to implementation. There are several CASE tools available for generating OMT diagrams, but, since it is not a standard but a methodology, the tools need to be embedded in a full standard such as ESA. OMT involves several stages, and is in some ways an extension to the Entity Relationship approach for designing and documenting systems. It is based strongly on the relationship between objects

and nouns in a textual description and on the behaviour of the object and the verbs to analyse the requirements.

Of widest use on the CICERO project have been the Object/Class model (a form of Extended Entity Relationship model showing static data relationships in the software), event traces (showing sequences of messages sent between objects to accomplish a given function) and the Dynamic Model (State Charts to define the temporal ordering of events impinging on a given object). These models have been supported by the use of a diagramming tool, Select. In addition, the CICERO project has used a code generation tool, OBLOG, which supports the OMT methodology, to generate User Components (in C++) from the OMT Object/Class model. The OBLOG tool integrates concepts from semantic data modelling and concurrent processing, employing objects as a unifying concept and aiming at a conceptually seamless methodology from requirements to implementation [15].

9 Experience of Applying Object Technology in Cortex

Having identified the technology and software engineering techniques used in the development of Cortex, conclusions made be drawn in their use.

9.1 Use of OMA standards

The main conclusions that can be drawn in Cortex from the experience of the use of OMA and CORBA are that:

• CORBA has successfully demonstrated the ability to allow the designers to wrap up existing legacy software systems, some of them shell scripts, as CORBA objects and integrate them into the Cortex system.

• the CORBA interface inheritance has allowed reuse of object behaviour.

• considering that the collaboration team is spread across the far corners of Europe and beyond with each group working on its own part of the prototype the “plugging” together of the components using CORBA to construct the prototype was a notable success.

• the CORBA tools (Orbix) have caused a few minor problems particularly time-out of object connections. However most of these problems have appeared to be resolved in the more recent version 1.3.

• none of the OMG fundamental object services (COSS) have been implemented as yet, in fact most of them have yet to be fully specified. This presents a dilemma as the CICERO team has proceeded to the design stage and have commenced specifying and designing its own object services. It will be interesting to see how long it takes for the OMG object services to be bundled into existing commercial CORBA Orbs.

• similarly the OMG Common Facilities Architecture has only just been published. However this document is intended more as a management tool intended for controlling development of and positioning Common Facilities and their specifications. The experience of using the OMG OMA within the CICERO Project may well provide the basis for the collaboration team to submit suggestions to the Common Facilities Task Force as to candidate common facilities within the control systems vertical market.

9.2 Use of ESA and OMT

The use of the ESA methods during the prototype development undoubtedly helped to clarify the design of the first release of Cortex. Such use of the ESA standards requires experience, especially when identifying the boundaries between the different phases of the lifecycle. In principle, ESA standards do not enforce any particular methodology. In practice, however, the ESA guidelines are clearly supporting a functional breakdown of the problem statement and

hence implicitly imply non object-oriented methodologies. Some trade-offs had to be made to support the OMT methodology, especially in the Architectural and Detailed Design phases of the ESA standards. One example is in resolving the relationship between the demands on the documentation produced in accordance with the ESA standards, particularly in the prominence given to natural language in the User and Software Requirements documents and the need for more precise semantics demanded by the OMT modelling approach. Guidelines are needed to help in the integration of these two standards.

Software produced for a pilot project is similar to production software with respect to robustness and reliability. Therefore management guidelines supporting the software lifecycle as enhanced by the ESA-PSS 05 standard must be strictly followed. In particular, as stated earlier, a complete set of Software Project Management, Software Configuration Management, Software Verification and Validation and Software Quality Assurance plans must be produced in parallel with the software lifecycle documents.

The OMT notation is strong in describing the abstract design but is lacking when describing the implemented system. This failing was felt during the Architectural and Detailed Design phases yet it is here, when the volume of design information increases substantially, that notations to capture the design and tools for their manipulation are essential. Work on design notations and the capture and re-use of designs is the focus of some researchers in the OO field and the results of this work should be made accessible to the CICERO team. OMT is not particularly targeted at the design of real-time systems and is not sufficient to handle concurrency. As a result, the treatment of some issues of concern in the development of Cortex are inadequately addressed. Some of these issues are better handled by later developments of the OMT approach, notably the Syntropy method [16]. Syntropy provides additional rigor to the OMT method, particularly in the design of cooperating processes.

It was clear during the prototype development phase that in normal discourse about Cortex, designers and users make great use of concrete examples of usage of the system and specific, telling, examples of message sequencing within the implemented system. It is felt that the efficiency of the design process and of communication of the design to the users could be improved by the incorporation of scenarios and use cases into the formal documentation and design process following the Jacobson approach [17].

The quality of tools support for methods remains an issue for the Cortex developers. Platform specific tools, difficult access to design repositories and the lack of support for multi-user and multi-site development, lead to a loss of efficiency. In particular, it generates additional conversion and cross-checking tasks. Managing versions of documents, supporting relationships between models from different viewpoints on the design and tracking the process of design decision making are all desirable in the selected tallest. As yet there is no obvious single tool for supporting the use of OMT in developing control systems. Varsamidis et al. [18] advocate the combination of individual software tools, each specialised in a particular aspect of the design process for control systems. Tool evaluation, selection and guidelines in their use in an integrated manner remain tasks for the next phase of the Cortex development.The use of CASE tools like OBLOG for the automated generation of CORBA compliant component code will be investigated further, especially to increase the generated codes performance, to deal with more complex data types and to support concurrent access.

10 Closing Comments

The CICERO project was approved as a CERN research and development project (RD-38) in February 1994. Since then the project has grown and continues to attract further commercial and academic research interest. Following the ESA standards, the Cortex element of CICERO has gone through preliminary User and Software Requirements specification followed by Architectural and Detailed Design [19] for a demonstrable prototype.

The Cortex approach of integrating processes in its Infrastructure, allows for the abstraction

of control system information through the encapsulation of the underlying system components. This provides for ease of interfacing between the active objects in a Cortex control system and facilitates the provision of standard software modules to perform the activities of communication and control. In addition, this abstraction enables the generation of such code automatically from the description of the system in the off-line Repository.

CICERO is not only addressing the re-use of existing control system software, it is also more widely addressing the reuse of existing control and automation facilities or functionalities. It offers a smooth Infrastructure to support scalability and partial software and system re-engineering.

Within a year, the CICERO collaboration has demonstrated that it was possible to build an heterogeneous distributed control application using Object Oriented techniques (OO programming languages, CORBA and OODBMS), commercial products and high level functionalities like alarm filtering, user assistance and on-line documentation to help the user operate the control system [20]. Much work remains in CICERO Phase II to provide a system which could be used in practice. This phase is expected to span a period of 22 months starting in March 1995. It is expected that CICERO Phase II will reach this goal and will prepare the ground for a final consolidation phase yielding a set of software building blocks to enable the implementation both of LHC experiment control systems and industrial complex control systems.

11 Acknowledgements

In a collaborative project such as CICERO RD38, each collaborator deserves the thanks of the authors. Rather than cite each contributor thanks are extended to members of RD38 from BARC (Bombay, India), CERN (Geneva, Switzerland), CIEMAT (Madrid, Spain), IVO International (Helsinki, Finland), KFKI (Budapest, Hungary), OBLOG (Lisbon, Portugal), SEFT (Helsinki, Finland), SpaceBel (Brussels, Belgium), UID (Linkoping, Sweden), USDATA (Dallas, USA), UWE (Bristol, UK), Valmet Automation (Tampere, Finland) and VTT (Oulu, Finland). Special thanks go to those involved with the development of Cortex.

12 References

[1] J. R. Rymer, ‘Common Object Request Broker - OMG’s New Standard for Distributed Object Management’, *Network Monitor* Vol. 6, No 9, pp 3-27 (1991)

[2] R. Barillere et al., ‘The Cortex Project: A Quasi- Real-Time Information System to Build Control Systems for High Energy Physics Experiments’, *NIM A* **352**, pp 492-496 (1994)

[3] L.R. Dalesio et al., ‘The Experimental Physics and Industrial Control System Architecture: Past, Present and Future’, *NIM A* **352**, pp 179-184 (1994)

[4] LHC: ‘The Large Hadron Collider Accelerator Project’, CERN AC 93-03 (1993).

[5] R. Barillere et al., ‘Ideas on a Generic Control System Based on the Experience of the Four LEP Experiments Control Systems’, Proceedings of the ICALEPCS ‘91 Conference, Tsukuba, Japan pp 246-253 (1991).

[6] OMG: ‘The Object Management Architecture (OMA Guide)’, Revision 2.0, Object Management Group Inc., OMG TC Document 91.11.1 (1992).

[7] CORBA: ‘Common Object Request Broker Architecture and Specification (CORBA)’, Revision 1.2, Object Management Group Inc., OMG TC Document 93.12.43 (1993).

[8] OMG: ‘Common Object Services Specification (COSS) Vol. 1’, Revision 1.0, Object Management Group Inc., First Edition (1994).

[9] OMG: ‘Common Facilities Architecture (CFA)’, Revision 4.0, Object Management Group

Inc., OMG TC Document 95-1-2 (1995).

[10] G. Filippi, 'Software Engineering for ESO's VLT Project', *NIM A* **352** pp 386-389 (1994).

[11] C. Mazza, 'Controlling Software Development', *NIM A* **352** pp 370-379 (1994).

[12] ESA: 'Guide to the Software Engineering Standards', ESA Board for Software Standardisation & Control (BSSC), (1991).

[13] B. W. Boehm, 'A Spiral Model of Software Development and Enhancement', IEEE Computer **21** (5), pp 61-72 (1988).

[14] J. Rumbaugh et al., *Object-Oriented Modelling & Design*, Prentice Hall, (1991).

[15] H-D. Ehrich, 'Fundamentals of Object-Oriented Information Systems Specification and Design: the OBLOG / TROLL Approach', *NIM A* **352** pp 375-378 (1994).

[16] S. Cook & J. Daniels, *Designing Object Systems - Object Oriented Modelling with Syntropy*, Prentice Hall (1994).

[17] I. Jacobson, *Object-Oriented Software Engineering - A Use Case Approach*, Addison-Wesley (1992).

[18] T. Varsamidis et al., 'Information Management for Control Systems Designers', Proceedings of the IEE International Conference on Control, Coventry, UK (1994).

[19] R. Barillere et al., 'Cortex User Requirements V2.1b', CERN RD-38/D/94-4-1 (1994, R. Barillere et al., 'Cortex Software Requirements V2', CERN RD-38/94-8-1 (1994), E. Bernard et al., 'Cortex System Design Document', SpaceBel Inc., SBI-CORTEX-ADD-001 & E.Bernard et al, 'Cortex Detailed Design Document', SpaceBel Inc., SBI-CORTEX-DDD-001

[20] G. Govindarajan et al., 'CICERO: 1994 Status Report', CERN RD-38/LHCC/95-15.